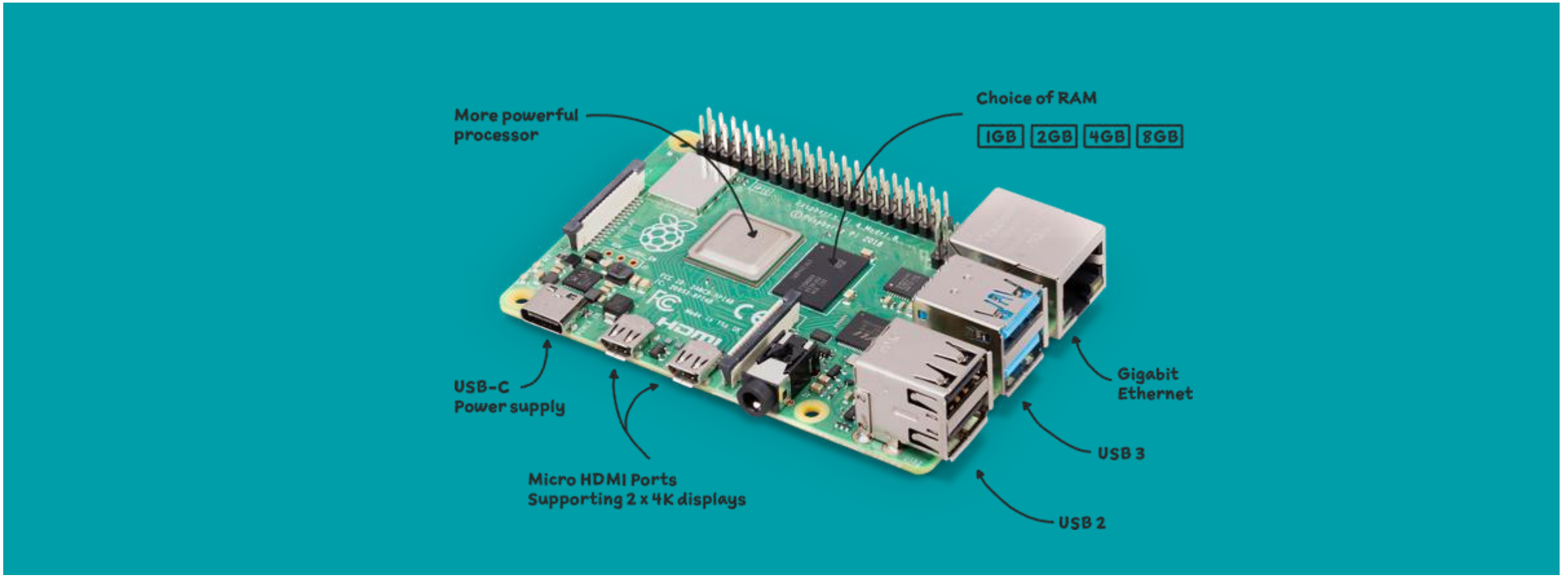
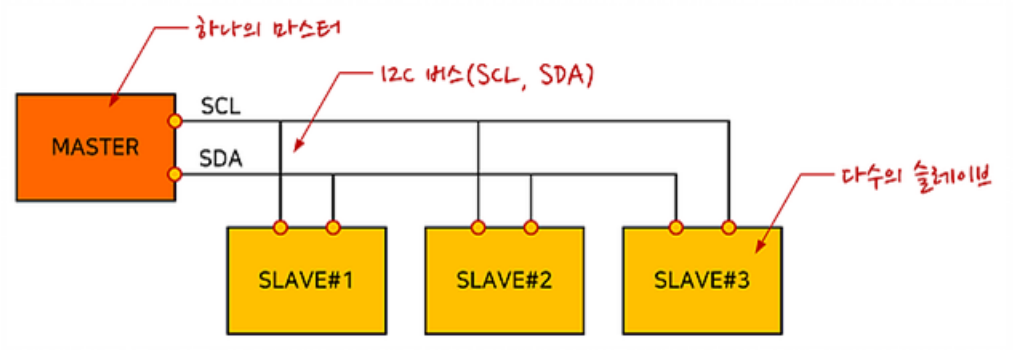


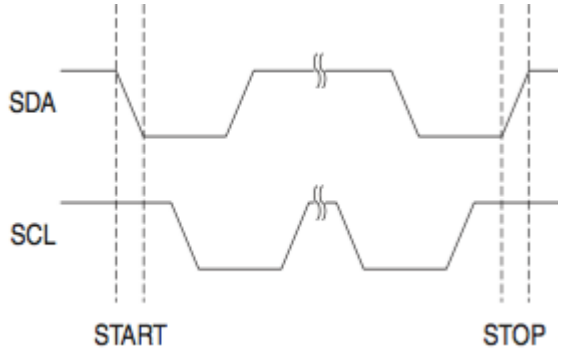
4. 센서활용-2

이영주
young.kopo@gmail.com

한국폴리텍대학

LED 제어(통신, I2C)





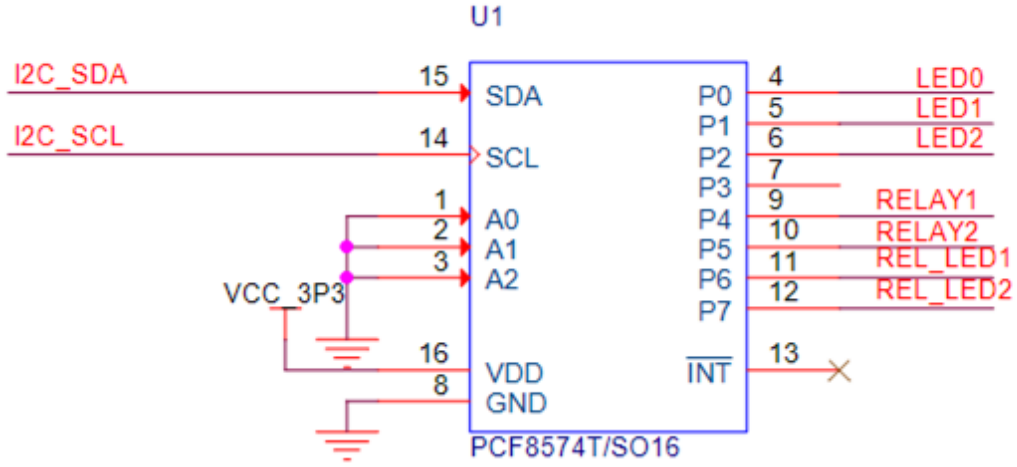
SCL : Serial Clock
SDA : Serial Data

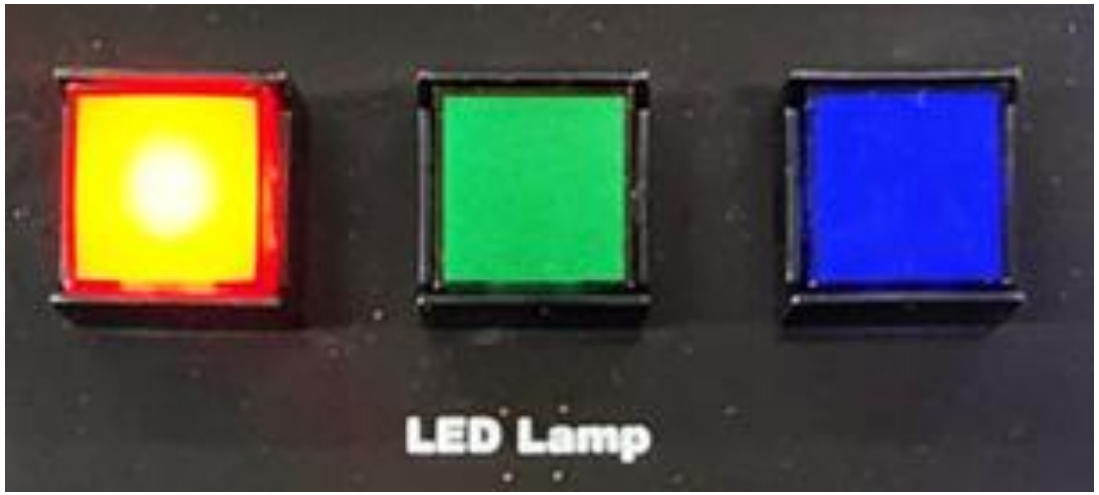
I2C(Inter Intergrated Circuit) 통신이란

- 두 개의 전선으로 여러 디바이스들을 연결할 수 있는 저속 통신 인터페이스

- 다른 통신 인터페이스에 비해 간단하며 한 개의 마스터(master)와 여러 개의 슬레이브(slave)들을 연결하여 SDA(Serial Data)와 SCL(Serial Clock) 두 개의 신호를 통해 데이터를 주고받음
- 송신과 수신이 동시에 불가능한 반이중(Half-Duplex) 방식,
- 각 슬레이브는 각자의 주소를 가지고 그 주소에 해당하는 슬레이브만 응답하여 데이터를 주고받음

LED 제어(통신, I2C)





PCF8574 Remote 8-Bit I/O Expander for I²C Bus

1 Features

- Low Standby-Current Consumption of 10 µA Max
- I²C to Parallel-Port Expander
- Open-Drain Interrupt Output
- Compatible With Most Microcontrollers
- Latched Outputs With High-Current Drive Capability for Directly Driving LEDs
- Latch-Up Performance Exceeds 100 mA Per JESD 78, Class II

2 Applications

- Telecom Shelters: Filter Units
- Servers
- Routers (Telecom Switching Equipment)
- Personal Computers
- Personal Electronics
- Industrial Automation
- Products with GPIO-Limited Processors

3 Description

This 8-bit input/output (I/O) expander for the two-line bidirectional bus (I²C) is designed for 2.5-V to 6-V V_{CC} operation.

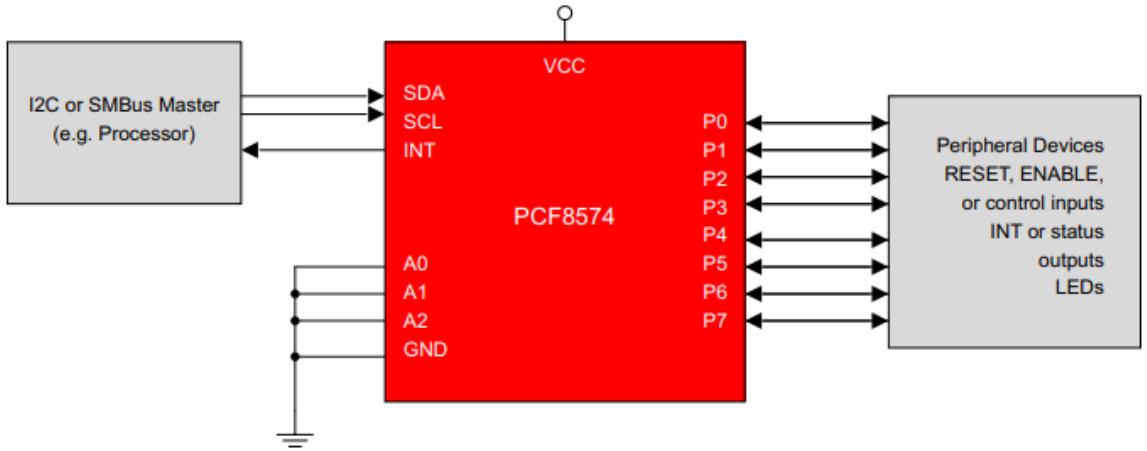
The PCF8574 device provides general-purpose remote I/O expansion for most microcontroller families by way of the I²C interface [serial clock (SCL), serial data (SDA)].

The device features an 8-bit quasi-bidirectional I/O port (P0–P7), including latched outputs with high-current drive capability for directly driving LEDs. Each quasi-bidirectional I/O can be used as an input or output without the use of a data-direction control signal. At power on, the I/Os are high. In this mode, only a current source to V_{CC} is active.

Device Information(1)

PART NUMBER	PACKAGE (PIN)	BODY SIZE (NOM)
PCF8574	TVSOP (20)	5.00 mm × 4.40 mm
	SOIC (16)	10.30 mm × 7.50 mm
	PDIP (16)	19.30 mm × 6.35 mm
	TSSOP (20)	6.50 mm × 4.40 mm
	QFN (16)	3.00 mm × 3.00 mm
	VQFN (20)	4.50 mm × 3.50 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.



LED 제어(통신, I2C) - 파이썬

파이썬 통신 모듈:

```
import smbus

bus = smbus.SMBus(1)

# I2C버스가 연결된  smbus 모듈 객체화
```

```
bus.write_byte(0x20, RED_LED)

#I2C 버스가 연결된 0x20  번지에 출력
```

```
import smbus
import time

RED_LED   = 0b00000001
GREEN_LED = 0b00000010
BLUE_LED  = 0b00000100

bus = smbus.SMBus(1)

# RED LED ON
bus.write_byte(0x20, RED_LED)
time.sleep(0.5)

# GREEN LED ON
bus.write_byte(0x20, GREEN_LED)
time.sleep(0.5)

# BLUE LED ON
bus.write_byte(0x20, BLUE_LED)
time.sleep(0.5)

state = RED_LED | GREEN_LED | BLUE_LED

# All LED ON
bus.write_byte(0x20, state)
time.sleep(0.5)

# Blue OFF
state = state & (~BLUE_LED)

# GREEN And RED LED ON
bus.write_byte(0x20, state)
```

LED 제어(통신, I2C)- 함수로

```
import smbus
import time

# LED 제어 비트
RED_LED   = 0b00000001
GREEN_LED = 0b00000010
BLUE_LED  = 0b00000100

state = None
bus = None

# i2c를 사용하기 위해 smbus 모듈을 초기화한다.
def ledInit():
    global state
    global bus
    state = 0b00000000
    bus = smbus.SMBus(1)

def ledOn(cmd):
    global state
    state = (state | cmd)
    bus.write_byte(0x20, state)

def ledOff(cmd):
    global state
    state = (state & (~cmd))
    bus.write_byte(0x20, state)
```

```
ledInit()

ledOn(RED_LED)
time.sleep(0.5)
ledOn(GREEN_LED)
time.sleep(0.5)
ledOn(BLUE_LED)
time.sleep(0.5)

ledOff(RED_LED)
time.sleep(0.5)
ledOff(GREEN_LED)
time.sleep(0.5)
ledOff(BLUE_LED)
time.sleep(0.5)

ledOn(RED_LED)
ledOn(GREEN_LED)
ledOn(BLUE_LED)
time.sleep(1)

ledOff(RED_LED)
ledOff(GREEN_LED)
ledOff(BLUE_LED)
time.sleep(1)
```

LED 제어(통신, I2C) - WiringPi

WiringPi 통신 함수:

```
#include <wiringPiI2C.h>

fd = wiringPiI2CSetup(0x20);

# I2C 0x20번지 초기화



wiringPiI2CWriteReg8(fd, 0x20, 0)

#I2C 버스가 연결된 0x20  번지에 출력
```

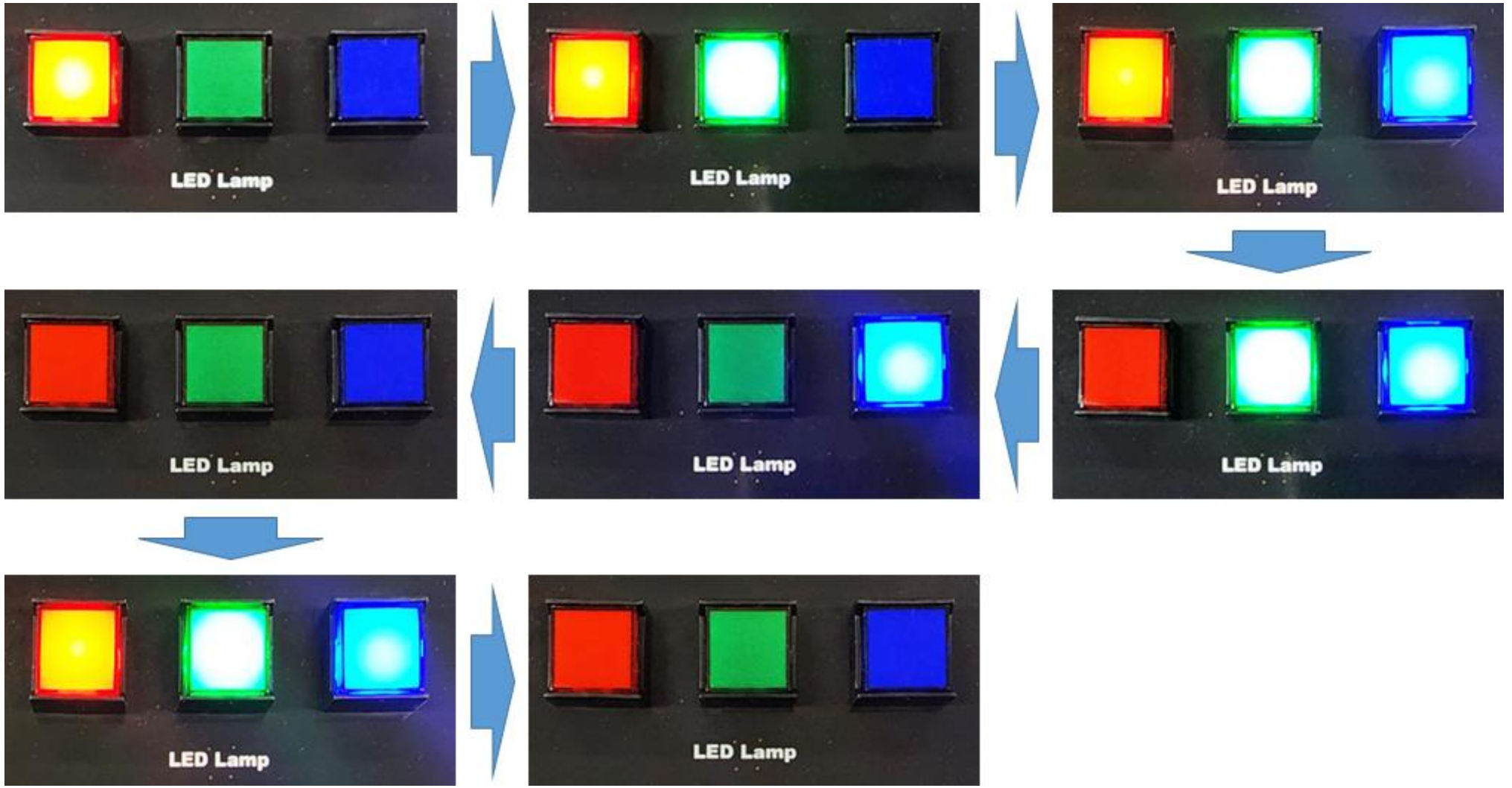
LED 제어(통신, I2C) -1

```c
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>

int main(void) {
    int ret,fd;
    char red_led = 0x01;
    char green_led = 0x02;
    char blue_led = 0x04;
    char state = red_led | green_led | blue_led;

    if (wiringPiSetup() == -1) {
        fprintf(stderr, "wiringPiSetup() is failed : %s₩n", strerror(errno));
        return 1;
    }

    fd = wiringPiI2CSetup(0x20);
    if (fd == -1) {
        fprintf(stderr, "wiringPiI2CSetup() is failed : %s₩n", strerror(errno));
        return 1;
    }
```

LED 제어(통신, I2C)-2

```
    if (wiringPiI2CWriteReg8(fd, 0x20, 0) == -1) {
        printf("write failed...₩n");
    }
    delay(1000);

    if (wiringPiI2CWriteReg8(fd, 0x20, state) == -1) {
        printf("write failed...₩n");
    }
    // write cycle time
    delay(5);

    return 0;
}
```

LED 제어(통신, I2C)- 순차 이동은?



Text LCD 제어(통신방식, I2C)

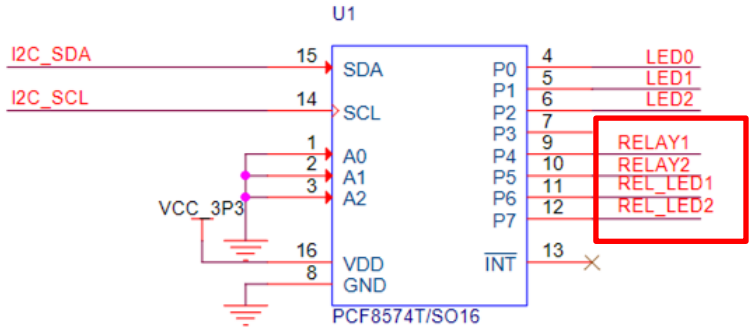
16개의 문자를 2줄로 표현
16*2 Character LCD 라고함

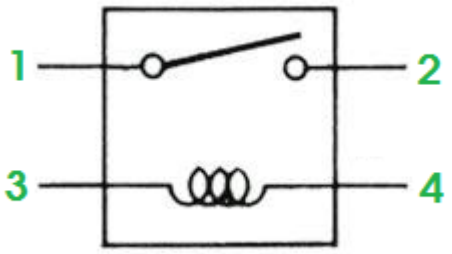
Text LCD 제어(통신방식, I2C)

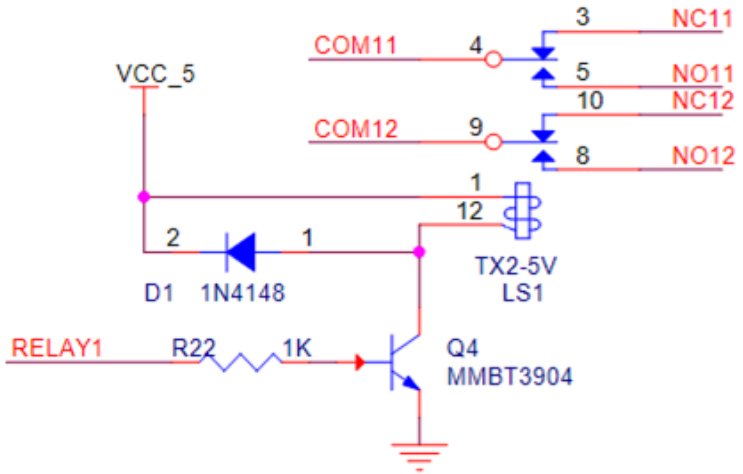
```
import I2C_LCD_driver

# I2C_LCD_Driver 모듈을 사용하는 예제
textLcd = I2C_LCD_driver.lcd()

# 16자 이내 아스키코드, 1은 첫 번째 줄이고 2는 두 번째
줄이다
textLcd.lcd_display_string("ABCDEFG", 1)
textLcd.lcd_display_string("123456", 2)
```

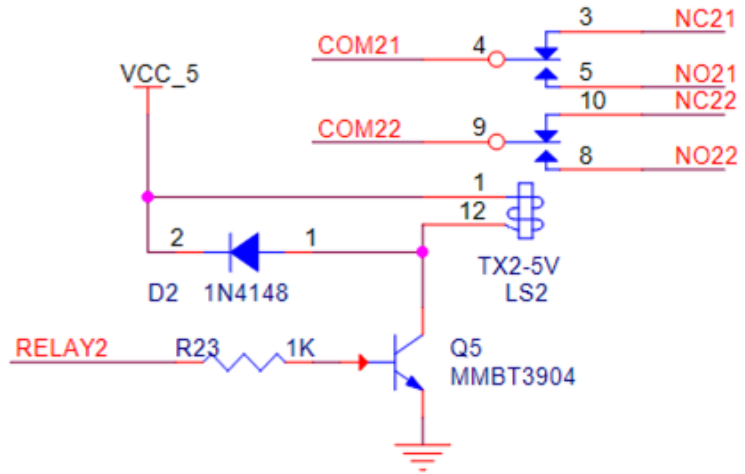
참고 https://gist.github.com/DenisFromHR/cc863375a6e19dce359d

Relay 제어(통신방식, I2C)









Relay 제어(통신방식, I2C)

```
import smbus
import time

RELAY_1    = 0b00010000
RELAY_2    = 0b00100000

bus = smbus.SMBus(1)

print(" 릴레이 1 ON / " + bin(RELAY_1))
bus.write_byte(0x20, RELAY_1)
time.sleep(1)

print(" 릴레이 2 ON / " + bin(RELAY_2))
bus.write_byte(0x20, RELAY_2)

# 현재 상태를 저장해서 OFF를 하기 위해 상태를 저장해놓는다.
state = RELAY_2
time.sleep(1)

state = state & (~RELAY_1)
print(" 릴레이 1 OFF / " + bin(state))
bus.write_byte(0x20, state)
time.sleep(1)

state = state & (~RELAY_2)
print(" 릴레이 2 OFF / " + bin(state))
bus.write_byte(0x20, state)
time.sleep(1)
```

Relay 제어(통신방식, I2C)

```
import smbus
import time

# Releay 제어 비트
RELAY_1    = 0b00010000
RELAY_2    = 0b00100000

state = None
bus = None

##########################
#    RELAY 제어 함수
##########################
# i2c를 사용하기 위해 smbus 모듈을 초기화한다.
def relayInit():
    global state
    global bus
    state = 0b00000000
    bus = smbus.SMBus(1)

def relayOn(cmd):
    global state
    # RELAY는 bit로 제어되기 때문에 논리 연산자를 사용한다.
    # |는 OR를 의미한다.
    state = (state | cmd)
    bus.write_byte(0x20, state)

def relayOff(cmd):
    global state
    # RELAY는 bit로 제어되기 때문에 논리 연산자를 사용한다.
    # not cmd 를 만들고 state와 and하여 원하는 비트만 OFF시킨다.
    state = (state & (~cmd))
    bus.write_byte(0x20, state)
```

```
relayInit()

# 릴레이1 ON/OFF
relayOn(RELAY_1)
time.sleep(0.5)
relayOff(RELAY_1)
time.sleep(0.5)

# 릴레이2 ON/OFF
relayOn(RELAY_2)
time.sleep(0.5)
relayOff(RELAY_2)
time.sleep(0.5)

# 릴레이1,2 ON
relayOn(RELAY_1)
relayOn(RELAY_2)
time.sleep(1)

# 릴레이1,2 OFF
relayOff(RELAY_1)
relayOff(RELAY_2)
time.sleep(1)
```

Relay 제어(통신방식, I2C)

```c
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>

#define RELAY_1 0x10
#define RELAY_2 0x20

char state = 0x0;
int fd;


void relay_init() {

if (wiringPiI2CWriteReg8(fd, 0x20, 0) == -1) {
        printf("write failed...₩n");
        }
        else printf("relay_init OK ₩n");

}

void relayOn(char cmd) {

state = state | cmd;

if (wiringPiI2CWriteReg8(fd, 0x20, state) == -1) {
printf("relay On failed...₩n");
}
else printf("relay_on OK[%x] ₩n", state);
}

void relayOff(char cmd) {
state = state & ~cmd;
     if (wiringPiI2CWriteReg8(fd, 0x20, state) == -1) {
        printf("relay On failed...₩n");
        }
        else printf("relay_off OK[%x] ₩n", state);
}
```

```c
int main(void) {

if (wiringPiSetup() == -1) {
fprintf(stderr, "wiringPiSetup() is failed : %s₩n", strerror(errno));
return 1;
}

fd = wiringPiI2CSetup(0x20);
if (fd == -1) {
fprintf(stderr, "wiringPiI2CSetup() is failed : %s₩n", strerror(errno));
return 1;
}

relay_init();

delay(500);
relayOn(RELAY_1);
delay(500);
relayOff(RELAY_1);
delay(500);

relayOn(RELAY_2);
delay(500);
relayOff(RELAY_2);
delay(500);


relayOn(RELAY_1 | RELAY_2);
delay(500);
relayOff(RELAY_1 | RELAY_2);
delay(500);

return 0;
}
```